Freie Universität Berlin

Volker Roth

Rechnersicherheit, SoSe 21

Übung 06

TutorIn: Oliver Wiese
Tutorium 02
Materialien: Latex, VSC, Skript

2. Juni 2021

1 Password Generation using Context Free Grammars

- *Consider the following probabilistic context-free grammar.*

Production rule	Probability
S $\rightarrow D_1L_3S_2D_1$	0.75
S $\rightarrow L_3D_1S_1$	0.25
$D_1 \rightarrow 4$	0.60
$D_1 \rightarrow 5$	0.20
$D_1 \rightarrow 6$	0.20
$S_1 \rightarrow !$	0.65
$S_1 \rightarrow \%$	0.30
$S_1 \rightarrow \#$	0.05
$S_2 \rightarrow \$\$$	0.70
$S_2 \rightarrow **$	0.30

- *and the following priority queue:*

Base Struct	Pre-Terminal	Probability	Pivot Value
$D_1^0L_3S_2^0D_1^0$	4L_3\$\$4	0.188	0
$L_3D_1^0S_1^0$	$L_3$4!	0.097	0

(a) *Calculate the next five pre-terminal structures that the enumerator (as discussed in class)does output and the resulting priority queue.*

1. After the first password was extracted:

Base Struct	**Pre-Terminal**	**Probability**	**Pivot Value**
$\mathrm{L}_3 D_1^0 S_1^0$	$\mathrm{L}_3 4!$	0.097	0
$\mathrm{D}_1^0 L_3 S_2^1 D_1^0$	$4\mathrm{L}_3 **4$	0.081	1
$\mathrm{D}_1^1 L_3 S_2^0 D_1^0$	$5\mathrm{L}_3\$\$4$	0.063	0
$\mathrm{D}_1^0 L_3 S_2^0 D_1^1$	$4\mathrm{L}_3\$\$5$	0.063	2

2. After the second password was extracted:

Base Struct	**Pre-Terminal**	**Probability**	**Pivot Value**
$\mathrm{D}_1^0 L_3 S_2^1 D_1^0$	$4\mathrm{L}_3 **4$	0.081	1
$\mathrm{D}_1^1 L_3 S_2^0 D_1^0$	$5\mathrm{L}_3\$\$4$	0.063	0
$\mathrm{D}_1^0 L_3 S_2^0 D_1^1$	$4\mathrm{L}_3\$\$5$	0.063	2
$\mathrm{L}_3 D_1^0 S_1^1$	$\mathrm{L}_3 4\%$	0.045	1
$\mathrm{L}_3 D_1^1 S_1^0$	$\mathrm{L}_3 5!$	0.0325	0

2 Code Review

Review and test the Python code of your peers. Your focus should be the implementation of the password-based authentication and consider at least the following aspects:

1. **Peer**:

   (a) *Used hash function and configuration.*

   Implemented

```
    def passwd(self, userid, passwd, flag="n"):
        salt = self.__getsalt()
        passwdhash = hashlib.md5((salt + passwd).encode('utf-8')).hexdigest()
```

   (b) *Usage and generation of salts and randomness.*

   Implemented

```
    def __getsalt(self):
        salt = ""
        length = random.randint(8, 15)
        for i in range(length):
            # get random char except ':'
            next_char = ':'
            while next_char == ':':
                next_char = chr(random.randint(33, 127))
            salt += next_char
        return salt
```

   (c) *Duplicate user names.*

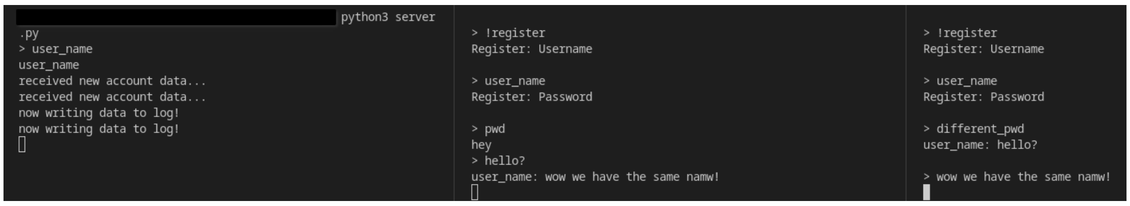
   Prevented

```
    def useradd(self, name, passwd):
        file = open(os.path.join(self.__path, self.__usrfile), "r+")
        for line in file:
            line = line.strip()
            content = line.split(':')
            if content[1] == name:
                file.close()
                return 1, []
        file.write(str(self.__idcount)+":"+name+":1\n")
        file.close()
        self.passwd(self.__idcount, passwd, "n")
        del passwd
        self.__idcount += 1
        return 0, [self.__idcount-1, 1]
```

(d) *Simultaneous creation of users with the same name.*

Not prevented. Still possible. I added a sleep in the server, just to demonstrate it, even without the sleep it can happen, as we work with threads. To fix this you can use locks in the server authentication method.

```python
    def useradd(self, name, passwd):
        file = open(os.path.join(self.__path, self.__usrfile), "r+")
        for line in file:
            line = line.strip()
            content = line.split(':')
            if content[1] == name:
                file.close()
                return 1, []
        print('received new account data...')
        time.sleep(10)
        print('now writing data to log!')
        file.write(str(self.__idcount)+":"+name+":1\n")
        file.close()
        self.passwd(self.__idcount, passwd, "n")
        del passwd
        self.__idcount += 1
        return 0, [self.__idcount-1, 1]
```



```
[redacted] python3 server
.py
> user_name
user_name
received new account data...
received new account data...
now writing data to log!
now writing data to log!
```

```
> !register
Register: Username

> user_name
Register: Password

> pwd
hey
> hello?
user_name: wow we have the same namw!
```

```
> !register
Register: Username

> user_name
Register: Password

> different_pwd
user_name: hello?

> wow we have the same namw!
```

(e) ***Extra:*** *I was able to find some security risks in your code with bandit.*

```
Test results:
>> Issue: [B303:blacklist] Use of insecure MD2, MD4, MD5, or SHA1 hash function.
   Severity: Medium   Confidence: High
   Location: user_manager.py:127
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html
    #b303-md5
126                 controlhash = content[2]
127                 passwdhash = hashlib.md5((salt + passwd).encode('utf-8')).
    hexdigest()
128                 break

--------------------------------------------------
>> Issue: [B303:blacklist] Use of insecure MD2, MD4, MD5, or SHA1 hash function.
   Severity: Medium   Confidence: High
   Location: user_manager.py:145
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html
    #b303-md5
144         salt = self.__getsalt()
145         passwdhash = hashlib.md5((salt + passwd).encode('utf-8')).hexdigest()
146         del passwd

--------------------------------------------------
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for
    security/cryptographic purposes.
   Severity: Low   Confidence: High
   Location: user_manager.py:190
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html
    #b311-random
189         salt = ""
190         length = random.randint(8, 15)
191         for i in range(length):

--------------------------------------------------
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for
    security/cryptographic purposes.
   Severity: Low   Confidence: High
   Location: user_manager.py:195
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html
    #b311-random
194             while next_char == ':':
195                 next_char = chr(random.randint(33, 127))
196             salt += next_char
```

- To fix the first two to security risks you could use the bcrypt module which has a 'safer' hashing function. Here is an example syntax:

```
import bcrypt
password = password.encode('utf-8')
hash_and_salt = bcrypt.hashpw(password_pepper, salt)
```

- To fix the third/fourth security risk you would need to create 'real' randomness, which can be done with a quantum PCs, but as you most likely don't have one right now, you can also use the bcrypt module to create a 'safer' salt. With this salt generator bandit will stop throwing an error. Here is the example:

```
salt = bcrypt.gensalt()
```

2. **Peer**:

   (a) *Used hash function and configuration.*

   Implemented

```python
pwd = hashlib.sha256(f"{SALT}{message}".encode('utf-8')).hexdigest()
```

   (b) *Usage and generation of salts and randomness.*

   Somewhat implemented. Salt is used but not random!

```python
SALT = 'MaRo'
```

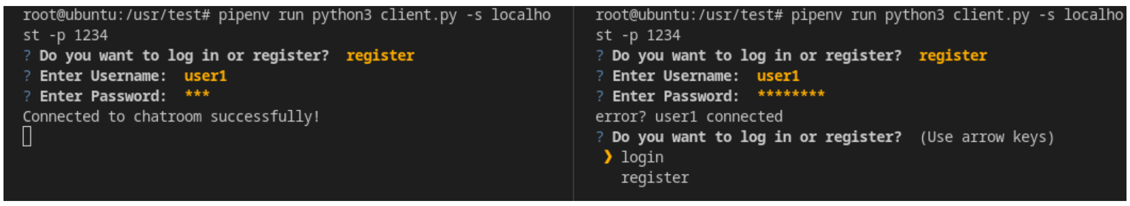
   (c) *Duplicate user names.*

   Prevented

```python
elif message == 'REGISTER':
    print('registration request received.')
    client_to_handle.send('USER'.encode())
    username = client_to_handle.recv(2048).decode()
    display_name = username
    pwd_entry = credential_table.get(username, None)
    if pwd_entry:
        client_to_handle.send('ALREADYTAKEN'.encode())
```

   (d) *Simultaneous creation of users with the same name.*

   Somewhat prevented. Might still be possible. But I could not test it, because the server does not allow it when multiple clients try to register / login at the same time. This seems more like a bug than a feature. If this would be fixed, simultaneous creation of users with the same name would be possible. To fix this you can use locks. Below is a example on how I tried to register with two clients at the same time. I added a print in the client to show the 'error'.



```
root@ubuntu:/usr/test# pipenv run python3 client.py -s localho
st -p 1234
? Do you want to log in or register?  register
? Enter Username:  user1
? Enter Password:  ***
Connected to chatroom successfully!
```

```
root@ubuntu:/usr/test# pipenv run python3 client.py -s localho
st -p 1234
? Do you want to log in or register?  register
? Enter Username:  user1
? Enter Password:  ********
error? user1 connected
? Do you want to log in or register?  (Use arrow keys)
 ❯ login
   register
```

   Here is the code snippet of you client. I added the lines 79 and 80.

```python
66 else:
67     server_socket.send('REGISTER'.encode())
68     resp = server_socket.recv(2048).decode()
69     if resp == 'USER':
70         server_socket.send(username.encode())
71         resp = server_socket.recv(2048).decode()
72         if resp == 'PW':
73             server_socket.send(password.encode())
74             resp = server_socket.recv(2048).decode()
75             if resp == 'OK':
76                 success = True
77         elif resp == 'ALREADYTAKEN':
78             print('username already taken, try another one.')
79         else:
80             print('error?', resp)
```

(e) ***Extra:*** *Bandit found no further security risky.*

(f) ***Errors:*** *I had some errors running your Dockerfile.*

- Your Dockerfile searched for 'client/client.py' and 'server/server.py', but they are named 'server\server.py' and 'server\server.py'. I fixed it by just renaming them. Here is the log file of the first time I ran your Dockerfile:

```
Sending build context to Docker daemon  133.1kB

Step 1/12 : FROM ubuntu:18.04
 ---> 81bcf752ac3d
Step 2/12 : ENV LANG C.UTF-8
 ---> Using cache
 ---> acaea361d2d2
Step 3/12 : ENV LC_ALL C.UTF-8
 ---> Using cache
 ---> 8a2ad5687918
Step 4/12 : RUN apt-get update && apt-get install -y python3.8     python3-pip
 ---> Using cache
 ---> 858c93f80f60
Step 5/12 : WORKDIR /usr/test
 ---> Using cache
 ---> 191d1b63cde9
Step 6/12 : COPY client/client.py .
COPY failed: file not found in build context or excluded by .dockerignore: stat
    client/client.py: file does not exist
```